EXHIBIT D

ActiveTables™ for mining data on the Internet

A wide variety of tabular data exists on the World Wide Web. This ranges from sports scorecards (NBA, NFL, etc.) to annual financial reports of companies. This document discusses, in brief, a mechanism for mining these data and providing additional interesting information about the data to the viewer, within the confines of a Java enabled web browser.

Problem Definition
A mechanism has to be developed whereby additional information can be obtained about the field(s) of a table that the user is viewing, and a way in which this information can be mined along with additional sources of information, to provide interesting patterns to the user. Ideally, this information should appear in a floating balloon, which constantly updates itself, as the user moves his/her mouse around in the table.

A table is represented in a web page as a series of HTML tags. Consequently, there is no way in which an application (other than the browser) can tell which column / row of the table the user's mouse is currently at. Also, there is no means, currently, of defining additional sources of information.

The ActiveTable™ mechanism is proposed to replace the current "passive" representation of tables through HTML tags.

What is an ActiveTable™ ?
An ActiveTable™ is an applet that contains the table to be mined. It is embedded in an HTML page. It is itself a JavaBean, and consists of the following JavaBeans :

(1) A set of **GridElements**. Each GridElement represents a single cell in the table. Each grid element "knows" its position (row/column) in the table, as well as the datum that it contains. Each GridElement is capable of responding to mouse movements over its visual representation in the ActiveTable™
(2) One or more **buttons**, e.g., to turn on/off data mining, to ask user defined questions, etc.
(3) A Floating **Balloon**, which would display interesting results from the current table. This updates continously, depending on the position of the user's mouse.
(4) A **text area**, which would display interesting data mining results and answers to user questions by combining the contents of the current table with other data from the web.
(5) Some other elements of pure "**decorative**" value, such as horizontal and vertical lines, bitmaps, etc.

Creating an ActiveTable™

An ActiveTable™ is a JavaBean, and can be inserted into a web page (?), or any other Java application, just like any other JavaBean, using a visual builder tool (such as IBM VisualAge, or Symantec Visual Café).

When a developer instantiates an ActiveTable™, a set of wizards appear, enabling him/her to customize the ActiveTable™. The wizards would require answers to questions such as the following :

(1) What is the **primary data source** of the ActiveTable™ ? (the primary data source of an ActiveTable™ refers to the source for the data actually displayed to the user by the ActiveTable, in its GridElements. This could be a subset of or an entire text file, ODBC table, etc). This information is **required.**

(2) What are the **secondary data sources,** if any ? (Secondary data sources refer to additional data sources that can provide more information to be mined. These could be text files, ODBC tables, text reports, video clips, etc.) This information is **optional.**

Once the primary data source for an ActiveTable™ has been identified, an applet representing it will be built by the builder tool, and inserted into the specified HTML page.

How will the ActiveTable™ work ?

When the user sees the HTML page, the ActiveTable™ applet will be invoked automatically by the browser. When she moves her mouse around on the table, the particular GridElement on which the mouse is located, will respond. The GridElement will perform some standard calculations, and update the floating balloon. For example, if the user's mouse is on the FGM column for Michael Jordan in the following table (row = 2, column = 2), the appropriate GridElement would respond.

Player	FGM	FGA
Jordan	5	10
Pippen	7	10
Kerr	6	8

Since the GridElement "knows" that it belongs to the 2nd row and 2nd column, it could display a message such as the following :

"Did you know that Michael Jordan made 5 out of the Bulls' 18 Field Goals (28 %) ?"

Alternatively, the GridElement could act more intelligently. Since it "knows" it belongs to the "FGM" column, it could query the "FGA" column, and come up with :

"Did you know that Michael Jordan shot 5 for 10 (50 %), whereas the Bulls shot 18 for 28 (64 %) ?"

Note that the first example could **always** work, in any domain, whereas the second represents a customization for a specific (in this case, basketball) domain.

When the user clicks on the Data Mining button, the ActiveTable™ would look up data from the secondary sources specified by the developer, and mine the results together. (The developer would have to provide some additional information while customizing the ActiveTable™, such as the Focus Attributes, etc.)

The user could click on the Ask Questions button, to ask her own questions. These questions would also be fired across all the (primary and secondary) data sources associated with the ActiveTable™

A typical DM / user question answer would be displayed in the answer text area, and would look like this : "Did you know that in this season, against the Knicks, in the games in which Jordan had more than 10 rebounds, the Bulls won 3 of 4 games (75%) "

Other Issues :

(1) The ActiveTable™ JavaBean can be converted into an ActiveX control, thus enabling the same functionality from within MS-Office containers, such as MS-Word, MS-Excel, etc.

(2) The ActiveTable™ could first be made available in its entirety to developers. The developer would be able to customize an ActiveTable™, by specifying different data sources, etc. Later, the various parts of the ActiveTable™ (buttons, text areas, etc.) could be made available individually, so that developers can customize the look and feel of the entire ActiveTable™ to their liking.

(3) The user can specify a text file / ODBC data sources as the secondary data source. In order to mine data, the current ActiveTable™ has to query / import data from these sources. This has various disadvantages, e.g., it assumes the existence of a database server (in case of ODBC), and it also involves querying, and sometimes, parsing files for importing data. However, if these secondary data sources are themselves represented using ActiveTables™ (i.e., they are the primary data sources of other ActiveTables™), then, using the RMI (Remote Method Invocation) mechanism available in Java, it might be possible to query these ActiveTables™ for secondary data, thus ensuring truly distributed computing. For example, consider the following scenario :

In the above example, the user clicks on data mining, while viewing Jordan's Field Goal Percentage. The ActiveTable™ issues a query to the NBA server, asking something like
"Any ActiveTables™ out there, who have data about the Bulls against the Knicks, please respond with some data (related to Shooting) for the game."

This query goes out to all ActiveTables™ on the NBA server, and evokes reponses from all the ActiveTables™ with primary data related to Bulls-Knicks games. The client ActiveTable™ can then use these data to perform some analysis, and show the results to the user.

Some comments on this mechanism :

a. It is totally independent of the way the data are stored on the server (text files, ODBC, etc.)
b. It does not overload any database server (however, it does cause additional burden to the HTTP server). From the point of view of security, a large company would prefer requests to the HTTP server rather than allowing millions of users direct access to their databases.
c. It involves minimal server-side programming (just setting up the mechanism by which ActiveTables™ could respond to these remote requests)
d. Data within the ActiveTable™ could be intelligently indexed for faster retrieval
e. Information obtained from various ActiveTables™ could be cached by the client ActiveTable™ for re-use
f. However, it does not make use of the database server's inherent optimization with respect to SQL queries.

Unclear at this point :

a. Once the ActiveTable™ has been customized by the developer, it can be saved to a ".ser" file. This will ensure that it retains all the properties that the developer has set. The ".ser" file can then be opened by a browser, like an applet, through an extension to HTML. However, as of now, no browser supports this serialized version of Java class files.
b. RMI requires more study to determine that all that has been mentioned above, is, in reality, possible to implement

Phase 1 of the project:

- Develop the beans for implementation of ActiveTable™.
- Implement the *customizer* for the ActiveTable™, where the ActiveTables™ is based on a single data source. (No filter for the data)
- Make the grids of the ActiveTable™ responds to mouse movements and pop up with simple but interesting statistical calculations on the data from the primary data source of the ActiveTable™.
- Start working on the re-design of the data mining algorithm, and the presentation of data mining results.

ActiveTable™ Design

The ActiveTable™ will appear as shown below :